Intensional properties of polygraphs

Guillaume Bonfante[1] **– Yves Guiraud**[2]

INRIA – LORIA[3]

January 8, 2006 – Modified October 2, 2007

***Abstract** – We present polygraphic programs, a subclass of Albert Burroni's polygraphs, as a computational model, showing how these objects can be seen as first-order functional programs. We prove that the model is Turing complete. We use polygraphic interpretations, a termination proof method introduced by the second author, to characterize polygraphic programs that compute in polynomial time. We conclude with a characterization of polynomial time functions and non-deterministic polynomial time functions.*

1   Introduction

Polygraphs are special higher-dimensional categories, introduced by Albert Burroni to provide a unified algebraic setting for rewriting [3]. For example, any term rewriting system can be translated into a polygraph which has, in case of left-linearity, exactly the same properties of termination and confluence [10, 6].

Here, we study how these mathematical objects can be used as a computational model. Informally, computations generated by a polygraph are done by a net of cells which individually behave according to some local transition rules. This model is close to John von Neumann's cellular automata [16] and Yves Lafont's interaction nets [9] with notable differences: while von Neumann's automata are essentially synchronous, interaction nets and polygraphs are asynchronous; polygraphs have a much more rigid geometry than interaction nets: the underlying graphs of the formers are directed acyclic graphs, preventing the "vicious circles" of the latters.

Termgraph rewriting systems provide another model of graphical computation [15]: it is an extension of term rewriting with an additional operation, sharing, that allows for a more correct representation of actual computation. The translation of terms into polygraphs is close to the one into termgraphs and they seem to have the same properties, as suggested by the first results in [8]. For example, let us consider the following term rewriting rule, used to compute the multiplication on natural numbers: $\mathtt{mult}(\mathtt{x}, \mathtt{succ}(\mathtt{y})) \rightarrow \mathtt{add}(\mathtt{x}, \mathtt{mult}(\mathtt{x}, \mathtt{y}))$. When applied, this rule duplicates the term corresponding to the argument $\mathtt{x}$. In termgraph rewriting, one is able to share it instead, so that there is no need for extra memory space. This sharing operation can be algebraically formalized as an operation with one input and two outputs, whose semantics is a duplication operation. In polygraphs, one can have many such operations with many outputs, explicitly represented and handled.

This is a key fact in our results on implicit computational complexity: indeed, the interpretations we consider here, called *polygraphic interpretations* [6, 8], can reflect the fact that two outputs of the same operation have some links between them, as we will see with the example of the list splitting function used in "divide and conquer" algorithms. This allows us to give complexity bounds where traditional

---

[1]guillaume.bonfante@loria.fr.

[2]yves.guiraud@loria.fr.

[3]UMR 7503 CNRS – INPL – INRIA – UHP – Nancy 2.

polynomial interpretations [13] cannot with the method described in [4, 1] or to give better bounds, as indicated here and in [8]. Moreover, the polygraphic interpretations give separated information on the spatial and on the temporal complexities of functions.

Because of space constraints, we only give the main arguments used to prove the results we present here; the complete proofs can be found in a longer paper, together with extended comments and technical details [2]. Let us note that the present paper proposes a slightly more general framework than the other one, allowing non-deterministic polygraphic programs.

In section 2 we introduce the notion of polygraphic program in an informal way and give the corresponding semantics we consider; we introduce the leading example of this paper, the polygraphic program computing the "fusion sort" on lists, and we prove that polygraphic programs form a Turing complete model of computation. In section 3, we recall the notion of polygraphic interpretation, give examples, define the notion of simple polygraphic program and prove results on termination of polygraphic programs. Finally, in section 4, we give polynomial complexity bounds for simple programs and prove that they characterize the classes PTIME and NPTIME of functions computable in polynomial time, respectively by a Turing machine and by a non-deterministic Turing machine.

2   Polygraphs as a computational model

The general definition of polygraph can be found in documents by Albert Burroni, Yves Lafont and François Métayer [3, 10, 14, 11, 12]. Here we give a rewriting-minded presentation of a special case of polygraphs, seeing them as rewriting systems on algebraic circuits.

Definition 2.1. A *monoidal 3-polygraph* is a composite object consisting of *cells*, *paths* and *compositions* organized into *dimensions*.

Dimension 1 contains elementary sorts called *1-cells* and represented by wires. Their concatenation $\star_0$ yields product types called *1-paths* and pictured as juxtaposed vertical wires. The empty product $*$ is also a 1-path, represented by the empty diagram.

Dimension 2 is made of operations called *2-cells*, with a finite number of typed inputs and outputs. They are pictured as circuit gates, with inputs at the top and outputs at the bottom. Using all the 1-cells and 2-cells as generators, one builds circuits called *2-paths*, using the following two compositions:

The constructions are considered *modulo* some relations, including topological deformation: one can stretch or contract wires freely, move 2-cells, provided one does not create crossings or break wires. Each 2-cell and each 2-path f has a 1-path $s_1(f)$ as input, its *1-source*, and a 1-path $t_1(f)$ as output, its *1-target*. The compact notation $f : s_1(f) \Rightarrow t_1(f)$ summarizes these facts.

Dimension 3 contains rewriting rules called *3-cells*. They always transform a 2-path into another one with the same 1-source and the same 1-target. Using all the 1-cells, 2-cells and 3-cells as generators, one can build reductions paths called *3-paths*, by application of the following three compositions, defined for F going from f to f' and G going from g to g': $F \star_0 G$ goes from $f \star_0 g$ to $f' \star_0 g'$; when $t_1(f) = s_1(g)$, then $F \star_1 G$ goes from $f \star_1 g$ to $f' \star_1 g'$; when $f' = g$, then $F \star_2 G$ goes from f to g'. These constructions

are identified *modulo* some relations, given in [7], where their 3-dimensional nature was explained. The relations allow one to freely deform the constructions in a reasonable way: in particular, they identify paths that only differ by the order of application of the same 3-cells on non-overlapping parts of a 2-path. A 3-path is *elementary* when it contains exactly one 3-cell. Each 3-cell and each 3-path F has a 2-path $s_2(F)$ as left-hand side, its *2-source*, and a 2-path $t_2(F)$ as right-hand side, its *2-target*. The notation $F : s_2(F) \Rightarrow t_2(F)$ stands for these facts.

For monoidal 3-polygraphs, rewriting notions are defined in a similar way as for term rewriting systems, with terms replaced by 2-paths, reduction steps by elementary 3-paths and reduction paths by 3-paths [6]. Hence, a *normal form* in a polygraph $\mathcal{P}$ is a 2-path f which is the 2-source of no elementary 3-path. The polygraph $\mathcal{P}$ *terminates* when it does not contain infinite families $(F_n)_{n \in \mathbb{N}}$ of elementary 3-paths such that $t_2(F_n) = s_2(F_{n+1})$ for all n. Other rewriting properties, such as *confluence* or *convergence* are also defined in an intuitive way.

Remark 2.2. As defined here, the k-paths of such a polygraph, equipped with their k compositions, form a k-category. More precisely, this is the k-category freely generated by all the i-cells, for $0 \leq i \leq k$ [3].

Definition 2.3. A *polygraphic program* is a monoidal 3-polygraph such that:

- Its 2-cells are divided into *structure 2-cells*, *constructors* and *functions*. The structure 2-cells consist of one $\bowtie : \xi \star_0 \zeta \Rightarrow \zeta \star_0 \xi$ for each pair of 1-cells (ξ, ζ), plus one $\triangle : \xi \Rightarrow \xi \star_0 \xi$ and one $\bullet : \xi \Rightarrow *$ for each 1-cell ξ. The constructors are 2-cells such with a 1-cell as 1-target. The functions are any 2-cells.

- Its 3-cells are divided between *structure 3-cells* and *computation 3-cells*. The structure 3-cells are given, for every constructor $\bigtriangledown : x \Rightarrow \xi$ and every 1-cell ζ, by:
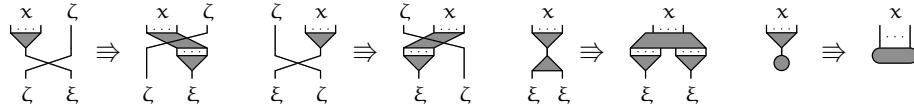
The 2-targets of the 3-cells of Δ_3^2 use structure 2-paths built from the structure 2-cells by using the following structural induction rules:
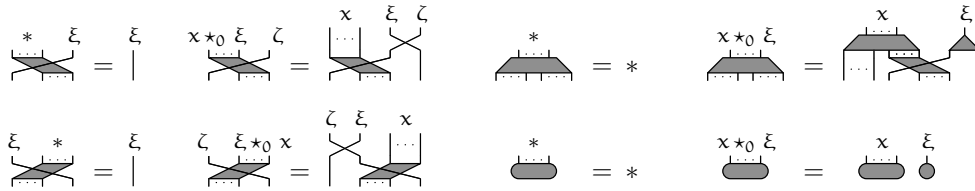
The computation 3-cells are 3-cells whose 2-source is of the shape $t \star_1 \varphi$, with φ a function 2-cell and t a 2-path built only with 1-cells and constructors. Furthermore, there is a finite constant that bounds the number of structure 2-cells in the 2-target of each computation 3-cell.

- For the present study, we assume that there exists a procedure to perform each step of computation: more formally, for every 3-path $F : f \rightarrow g$ containing exactly one 3-cell, the map giving g from the pair (f, F) is computable in polynomial time.

Example 2.4. We consider the following polygraphic program with one 1-cell, two constructors φ and ϕ, two functions $\triangledown$ and $\blacktriangledown$ and four computation 3-cells (we do not give the structure cells):

$$\varphi\!\!\triangledown \Rightarrow \mid \qquad \phi\!\!\triangledown \Rightarrow \triangledown \qquad \varphi\!\!\blacktriangledown \Rightarrow {}^{\bullet}_{\circ} \qquad \phi\!\!\blacktriangledown \Rightarrow \triangle\!\!\blacktriangledown$$

With the constructors, one can represent the natural number n, using φ for 0 and ϕ for the successor operation, yielding a 2-path t_n with zero input and one output. Furthermore, one can check that this polygraph is convergent and that, given t_m and t_n, the normal form of $(t_m \star_0 t_n) \star_1 \triangledown$ is t_{m+n}, while the one of $(t_m \star_0 t_n) \star_1 \blacktriangledown$ is t_{mn}.

Hence this polygraphic program computes the addition and the multiplication on natural numbers: the 1-cells are the data types, the 2-paths $\xi \Rightarrow *$ built only from constructors are the values, while the result of the application of a function $\triangledown$ with n inputs to well-typed values $(t_1, \ldots, t_n)$ is the normal form of the 2-path $(t_1 \star_0 \cdots \star_0 t_n) \star_1 \triangledown$. This semantical interpretation is formalized thereafter.

Definition 2.5 (Semantics). Let us fix a polygraphic program $\mathcal{P}$. If ξ is a 1-cell, a *term of type* ξ is a 2-path built only with constructors and with ξ as 1-target. A *value* or *closed term* is a term with no input. The set of values with type ξ is denoted by $\mathcal{V}(\xi)$. The *domain of computation* of $\mathcal{P}$ is the multi-sorted algebra made of the family of all the sets $\mathcal{V}(\xi)$ equipped with the operations given, for each constructor $\gamma : \xi_1 \star_0 \cdots \star_0 \xi_n \Rightarrow \xi$, by the map still denoted by γ:

$$\gamma : \mathcal{V}(\xi_1) \times \cdots \times \mathcal{V}(\xi_n) \to \mathcal{V}(\xi)$$
$$(t_1, \ldots, t_n) \mapsto (t_1 \star_0 \cdots \star_0 t_n) \star_1 \gamma.$$

Let us consider a function f from $\mathcal{V}(\xi_1) \times \cdots \times \mathcal{V}(\xi_m)$ to $\mathcal{V}(\zeta_1) \times \cdots \times \mathcal{V}(\zeta_n)$. There are two cases whether $\mathcal{P}$ is confluent or not.

If $\mathcal{P}$ is confluent, we say that $\mathcal{P}$ *computes* f if there exists a 2-path, still denoted by f, from $\xi_1 \star_0 \cdots \star_0 \xi_m$ to $\zeta_1 \star_0 \cdots \star_0 \zeta_n$, such that, for every family $(t_1, \ldots, t_m)$ of values in $\mathcal{V}(\xi_1) \times \cdots \times \mathcal{V}(\xi_m)$, the 2-path $(t_1 \star_0 \cdots \star_0 t_m) \star_1 f$ normalizes into the family $f(t_1, \ldots, t_m)$ of values in $\mathcal{V}(\zeta_1) \times \cdots \times \mathcal{V}(\zeta_n)$. Let us note that the normal form is unique in that case.

If $\mathcal{P}$ is not confluent, after Gurevich and Grädel [5], we say that it computes f if, for all values $(t_1, \ldots, t_m)$ in $\mathcal{V}(\xi_1) \times \cdots \times \mathcal{V}(\xi_m)$, the following holds, where the considered order is the lexicographic order:

$$f(t_1, \ldots, t_m) = \max \left\{ \text{normal forms of } (t_1 \star_0 \cdots \star_0 t_m) \star_1 f \right\}.$$

Example 2.6. Let us consider a polygraphic program that computes, among other functions, the *fusion sort* function on lists of natural numbers. It has two 1-cells, nat for natural numbers and list for lists of natural numbers. Its other cells, apart from structure ones are:

- Constructors: one $\textcircled{n} : * \Rightarrow$ nat for each natural number n, plus $\varphi : * \Rightarrow$ list for the empty list and $\triangledown : $ nat $\star_0$ list $\Rightarrow$ list for the list constructor.

- Functions: the main $\bullet : $ list $\Rightarrow$ list for fusion sort, together with $\blacktriangle : $ list $\Rightarrow$ list $\star_0$ list for splitting lists and $\blacktriangledown : $ list $\star_0$ list $\Rightarrow$ list for merging them.
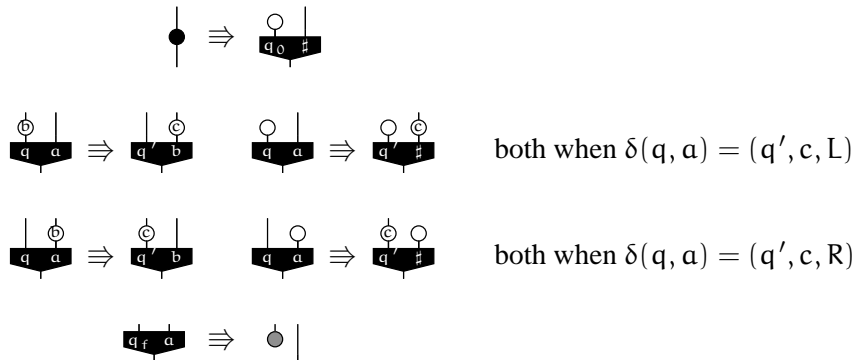
- Computation 3-cells:
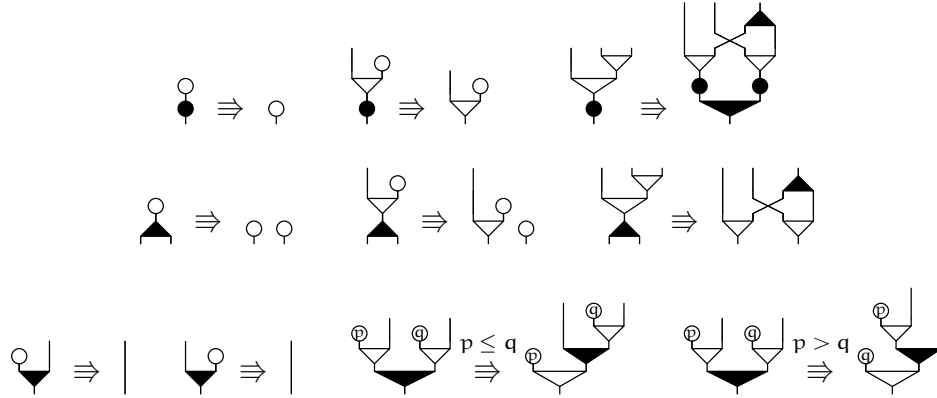
Note that the last two rules for the ▼ function are, in fact, a notation for an infinite family of 3-cells: there is exactly one of them for each pair (p, q) of natural numbers, depending which one of $p \leq q$ or $p > q$ holds. However, these two conditions are computable (in linear time), preventing super-Turing computations. We have chosen a simplified representation of natural numbers which considers them as being predefined, at the "hardware level", together with their predicate $\leq$. The reason for this choice is to postpone the study of modularity and of the `if-then-else` construction to subsequent work.

Theorem 2.7. *Polygraphic programs form a Turing-complete model of computation.*

Proof. Here we give a sketch of the proof, while the complete one can be found in [2]. We could use the fact that we can simulate rewriting systems which are Turing complete, but we give the explicit simulation as a first step to the proof of Theorem 4.9. Given a Turing machine, one defines:

- Constructors: one $\bigcirc : 0 \Rightarrow 1$ for the empty word plus one $\varphi : 1 \Rightarrow 1$ for each letter a.

- Functions: one $\bullet : 1 \Rrightarrow 1$ for the function to be computed plus one $\mathtt{step}_{q,a} = \boxed{q \ a} : 2 \Rightarrow 1$ for each state q and each letter a, including the blank symbol $\sharp$.

- Computation 3-cells are given thereafter, the first rule initializing the computation, the four subsequent families replicating the transitions of the Turing machine and the final family starting the computation of the result:

both when $\delta(q, a) = (q', c, L)$

both when $\delta(q, a) = (q', c, R)$

We assume that, at the end of the computation, the result of the Turing machine is the word written on the tape at the right of the head: this is why the last 3-cell erases the left part.

Let us assume that the machine is in the state q, reading the letter a, with w_l and w_r the two words respectively written on the left of a, from right to left, and on the right of a, from left to right. Then, this state of the system is represented by the 2-path $(w_l \star_0 w_r) \star_1$. Then one checks that each transition step of the Turing machine corresponds to an elementary 3-path of its polygraphic version. □

3 Polygraphic interpretations and simple programs

Polygraphic interpretations have been introduced to prove termination of 3-polygraphs [6]. Here we use a restricted form to get properties on the complexity of polygraphic programs. In order to give some intuition, let us imagine that 2-paths are electrical circuits, crossed by currents going downwards, from the inputs to the outputs.

A polygraphic interpretation associates to each 2-cell φ a "current map" φ_* and a "heat map" $[\varphi]$. The current map gives the intensity of the currents leaving the circuit gate φ, according to the intensity of the incoming currents. The heat map says how much heat is produced by the same gate φ during this process.

From these maps, one can compute the currents and heat produced by each 2-path. The original use was to find a polygraphic interpretation such that each reduction step replaces a 2-path by another one that produces strictly less heat. Here, the current maps will be used as approximations of the size of the computed values, while heat maps will estimate the number of computation steps remaining to reach a result: hence, current maps and heat maps will give bounds respectively on the spatial and on the temporal complexities of a polygraphic program.

Definition 3.1. A *polygraphic interpretation* of a polygraphic program $\mathcal{P}$ consists into a mapping of each 2-path f with m inputs and n outputs onto two monotone maps $f_* : \mathbb{N}^m \to \mathbb{N}^n$ and $[f] : \mathbb{N}^m \to \mathbb{N}$, such that the following conditions are satisfied:

- For every 1-path x of length n, we have $x_* = \mathrm{Id}_{\mathbb{N}}^n$ and $[x] = 0$.

- For every 2-paths f and g, the following equalitities hold when defined:

  - $(f \star_0 g)_*(\vec{x}, \vec{y}) = (f_*(\vec{x}), g_*(\vec{y}))$ and $[f \star_0 g](\vec{x}, \vec{y}) = [f](\vec{x}) + [g](\vec{y})$;
  - $(f \star_1 g)_*(\vec{x}) = g_*(f_*(\vec{x}))$ and $[f \star_1 g](\vec{x}) = [f](\vec{x}) + [g](f_*(\vec{x}))$.

Given an interpretation and a 2-cell φ, we denote by φ_*^j the jth component of the map φ_*. An interpretation of $\mathcal{P}$ generates a binary relation denoted by $\succ$: it is defined, on 2-paths f and g with the same 2-source and the same 2-target, by $f \succ g$ when the two inequalities $f_*(\vec{\imath}) \geq g_*(\vec{\imath})$ and $[f](\vec{\imath}) > [g](\vec{\imath})$ hold for every possible family $\vec{\imath}$ of natural numbers. An interpretation is *compatible* with a 3-cell α when $s_2(\alpha) \succ t_2(\alpha)$ and *weakly compatible* with α if $s_2(\alpha) \succeq t_2(\alpha)$.

It was proved in [6] that an interpretation is entirely determined by its values on the 2-cells of the polygraph, that the binary relation $\succ$ is a terminating strict order and that context are strictly monotone with respect to it. These are the main steps towards the following result.

Theorem 3.2 ([6])**.** *If a polygraphic program admits an interpretation which is compatible with all of its 3-cells, then it terminates.*

Example 3.3. Let us assume that we have a current map $(\cdot)_*$ on a polygraphic program such that the following conditions hold:

- If ▽ is a constructor with n inputs, then $▽_*(i_1, \ldots, i_n) > i_1 + \cdots + i_n$.

- One structure 2-cells, we have $⋈_*(i, j) = (j, i)$ and $△_*(i) = (i, i)$.

We define a heat map $[\cdot]_S$ as follows:

- If ▽ is a constructor or a function, then $\left[▽\right]_S = 0$.

- On structure 2-cells, we have $\left[⋈\right]_S(i, j) = ij$, $\left[△\right]_S(i) = i^2$ and $\left[⬤\right]_S(i) = i$.

It is proved in [2] that these values generate a polygraphic interpretation compatible with the structure 3-cells. Hence theorem 3.2 tells us that a polygraphic program without computation 3-cell terminates.

Definition 3.4. Given a current map $(\cdot)_*$ on a polygraphic program that satisfies the conditions of example 3.3, the heat map $[\cdot]_S$ is called *structure heat* generated by $(\cdot)_*$.

Definition 3.5. We denote by $\mathbb{N}[x_1, \cdots, x_n]$ the algebra of polynomials in n variables and coefficients in $\mathbb{N}$. Let $\mathcal{P}$ be a polygraphic program. A polygraphic interpretation is *simple* when the following conditions are met:

- For any 2-cell φ with m inputs and n outputs, the maps $\sum_{j=1}^{n} \varphi_*^j$ and $[\varphi]$ are in $\mathbb{N}[x_1, \ldots, x_m]$.

- If γ is a constructor with n inputs, then $\gamma_* = \sum_{i=1}^{m} x_i + a_\gamma$, with $a_\gamma > 0$, and $[\gamma] = 0$. Moreover, we assume that there exists a $a \in \mathbb{N}^*$ bounding all the a_γ's.

- On structure 2-cells, one has $⋈(i, j) = (j, i)$ and $△(i) = (i, i)$. Moreover, structure cells produce no heat: $\left[△\right](i) = 0$, $\left[⋈\right](i, j) = 0$, $\left[⬤\right](i) = 0$.

- For every function φ with m inputs and n outputs and for every family $(i_1, \ldots, i_m)$ of natural numbers, we have $\sum_{j=1}^{n} \varphi_*^j(i_1, \ldots, i_m) \geq i_1 + \cdots + i_m$.

A polygraphic program is called *simple* when the 2-targets of its computation 3-cells contain at most K structure 2-cells, for some fixed K, and when it admits a simple polygraphic interpretation which is compatible with all of its computation 3-cells.

Theorem 3.6. *A simple polygraphic program terminates.*

Proof. Let $\mathcal{P}$ be a simple polygraphic program and let $(\cdot)_*$ and $[\cdot]$ be the current and heat maps of a simple interpretation, compatible with all the computation 3-cells of $\mathcal{P}$. It is a direct computation to check that such an interpretation is weakly compatible with the structure 3-cells of $\mathcal{P}$. Hence, we deduce that $\mathcal{P}$ terminates if and only if the polygraphic program $\mathcal{Q}$ does, where $\mathcal{Q}$ is built from $\mathcal{P}$ by removal of the computation 3-cells. The map $(\cdot)_*$ also satisfies the conditions to generate a structure heat map $[\cdot]_S$ proving the termination of $\mathcal{Q}$. □

Example 3.7. Let us prove that the polygraphic program of example 2.6 is simple. Let us consider the interpretation generated by these values:

- $\Phi_* = 1$, $\quad Q_* = 1$, $\quad \bigtriangledown_*(i,j) = i + j + 1$;

- $\bullet_*(i) = i$, $\quad \blacktriangle_*(i) = (\lceil i/2 \rceil, \lfloor i/2 \rfloor)$, $\quad \blacktriangledown_*(i,j) = i + j$;

- $[\bullet](i) = 2i^2$, $\quad [\blacktriangle](i) = i$, $\quad [\blacktriangledown](i,j) = i + j$.

We have used the notations $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ for the rounding functions, respectively by excess and by default. This interpretation meets the conditions of definition 3.5 and, thus, is simple. Now, one has to check that it is compatible with all the computation 3-cells: we give some of the computations for the last 3-cell of the function $\bullet$. Let us start with $(\cdot)_*$. On one hand:

$$\left(\vcenter{\hbox{\includegraphics{}}} \right)_* (i,j,k) = \left(\vcenter{\hbox{\includegraphics{}}} \right)_* (i, \bigtriangledown_*(j,k)) = \bullet_* \circ \bigtriangledown_* (i, \bigtriangledown_*(j,k)) = i + j + k + 2.$$

And, on the other hand:

$$\left(\vcenter{\hbox{\includegraphics{}}} \right)_* (i,j,k) = i + j + \lceil k/2 \rceil + \lfloor k/2 \rfloor + 2 = i + j + k + 2.$$

Now, let us consider $[\cdot]$. For the 2-source of the 3-cell, one gets:

$$\left[\vcenter{\hbox{\includegraphics{}}} \right] (i,j,k) = [\bullet](i + j + k + 2) = 2 \cdot (i + j + k + 2)^2.$$

And, for its 2-target, $\left[\vcenter{\hbox{\includegraphics{}}} \right] (i,j,k)$ is equal to:

$$[\blacktriangle](k) + [\bullet](i + \lceil k/2 \rceil + 1) + [\bullet](j + \lfloor k/2 \rfloor + 1) + [\blacktriangledown](i + \lceil k/2 \rceil + 1, j + \lfloor k/2 \rfloor + 1)$$
$$= 2 \cdot (i + \lceil k/2 \rceil + 1)^2 + 2 \cdot (j + \lfloor k/2 \rfloor + 1)^2 + i + j + 2k + 2.$$

We conclude, for example, by considering two cases, depending on the parity of k.

Example 3.8. For the polygraphic program of example 2.4, the following values generate a simple interpretation which is compatible with the four computation 3-cells:

- $Q_* = 1$, $\quad \Phi_*(i) = i + 1$, $\quad \blacktriangle_*(i) = (i,i)$, $\quad \bigtriangledown_*(i,j) = i + j$, $\quad \blacktriangledown_*(i,j) = ij$;

- $[Q] = [\Phi](i) = [\blacktriangle](i) = [\bullet](i) = 0$, $\quad [\bigtriangledown](i,j) = i$, $\quad [\blacktriangledown](i,j) = (i + 1)j$.

4 Complexity of simple programs

Definition 4.1. Let $\mathcal{P}$ be a polygraphic program. If f is a 2-path of $\mathcal{P}$, we denote by $\|f\|$ the number of 2-cells f is made of. If F is a 3-path of $\mathcal{P}$, we denote by $\||F\||$ the number of 3-cells F is made of.

Let $\mathcal{P}$ be a simple program with a fixed interpretation made of $(\cdot)_*$ and $[\cdot]$. We want to prove that $(\cdot)_*$ is a good estimation of the size of values computed by $\mathcal{P}$, given by $\|\cdot\|$, while $[\cdot]$ is one for the size of the computations, given by $\||\cdot\||$. Once again, the complete proofs are in [2]. We recall that, by assumption, there exists a $a > 0$ that bounds each $a_\gamma = \gamma_*(0, \ldots, 0)$, for γ a constructor. By induction on the size of values, we prove that $(\cdot)_*$ is an estimation of the size of values:

Lemma 4.2. *For every value* t, *the inequalities* $\|t\| \leq t_* \leq a\,\|t\|$ *hold in* $\mathbb{N}$.

Using the properties of the polygraphic interpretation we consider and lemma 4.2, we prove that the size of intermediate and of final values are bounded by a polynomial in the size of the initial values:

Proposition 4.3. *Let* φ *be a function with* m *inputs and* n *outputs. Let* P_φ *be the polynomial in* $\mathbb{N}[x_1, \ldots, x_m]$ *defined by* $\mathsf{P}_\varphi = \sum_{j=1}^{n} \varphi_*^j(ax_1, \ldots, ax_m)$. *Let* t *be a family of values of type* $s_1(\varphi)$ *and let us assume that* $t \star_1 \varphi$ *reduces into a 2-path of the shape* $u \star_1 c$, *where* u *has* p *outputs. Then the inequality* $\sum_{j=1}^{p} u_*^j \leq \mathsf{P}_\varphi(\|t^1\|, \ldots, \|t^m\|)$ *holds. In particular, if* $u = \varphi(t)$, *the inequality* $\|\varphi(t)\| \leq \mathsf{P}_\varphi(\|t^1\|, \ldots, \|t^m\|)$ *holds.*

Example 4.4. If one computes these polynomials for the simple polygraphic program of example 2.6, one sees that, for any list t, the sorted list $\blacklozenge(t)$ and all the intermediate values computed to reach the result have their sizes bounded by the size of t:

$$P_{\blacklozenge}(x) = \blacklozenge_*(1 \cdot x) = x, \qquad P_{\blacktriangledown}(x, y) = \blacktriangledown_*(1 \cdot x, 1 \cdot y) = x + y,$$

$$P_{\blacktriangle}(x) = \blacktriangle_*^1(1 \cdot x) + \blacktriangle_*^2(1 \cdot x) = \lceil x/2 \rceil + \lfloor x/2 \rfloor = x.$$

For the polygraphic program of example 2.4, one gets $P_{\blacktriangledown}(x, y) = x + y$ and $P_{\blacktriangledown}(x, y) = xy$. Hence, the current maps give us information on the spatial complexity of the computation, separated from the length of computations.

Now we interest ourselves into polynomial bounds for the length of computations. We start by a technical lemma, which proves that, during a computation, the potential structure heat increase due to the application of a computation 3-cell is polynomially bounded by the size of the arguments. We recall that, by assumption, each computation 3-cell contains at most K structure 2-cells.

Lemma 4.5. *Let* φ *be a function with* m *inputs. We denote by* S_φ *the polynomial* $K \cdot P_\varphi^2$. *Let* t *be a family of values of type* $s_1(\varphi)$, *let* f *and* g *be 2-paths such that* $t \star_1 \varphi$ *reduces into* f *which itself reduces into* g *by application of a computation rule* α. *Then the following inequality holds:*

$$[f]_S + S_\varphi(\|t^1\|, \ldots, \|t^m\|) \geq [g]_S.$$

Proof. The complete, technical proof is in [2]. Here we recall the main reasoning steps. We denote by $\alpha : a \Rightarrow b$ the computation 3-cell used to reduce f into g. We decompose f and g to make a and b appear and use the properties of current and heat maps to conclude that there exist natural numbers $i_1, \ldots, i_m$

such that the inequality $[f]_S + [b]_S(i_1, \ldots, i_m) \geq [g]_S$ holds. Then we prove that $[b]_S(i_1, \ldots, i_m)$ is polynomially bounded by the size of t. By definition of the structure heat, $[b]_S(i_1, \ldots, i_m)$ is the sum of all the structure heats produced by the structure 2-cells b is made of. Then we use proposition 4.3 to prove that the current incoming in each input of each structure 2-cell of b is bounded by $P_\varphi(\|t^1\|, \ldots, \|t^m\|)$. Then, by definition of $[\cdot]_S$ on structure 2-cells, we conclude that the structure heat produced by each one is at most $P_\varphi^2(\|t^1\|, \ldots, \|t^m\|)$. Finally, we use the fact that b is the 2-target of a computation 3-cell to deduce that there is at most K structure 2-cells in b. $\qquad \square$

Example 4.6. For the polygraphic program of example 2.6 we have $K = 1$, $S_\bullet(x) = x^2$, $S_\blacktriangle(x) = x^2$ and $S_\blacktriangledown(x, y) = (x + y)^2$. For the one of example 2.4, we have $K = 1$, $S_\blacktriangledown(x, y) = (x + y)^2$ and $S_\blacktriangledown(x, y) = x^2 y^2$.

Now let us prove that the length of a computation is polynomially bounded by the size of the arguments.

Proposition 4.7. *Let φ be a function with m inputs. We define the following polynomials:*

$$Q_\varphi(x_1, \ldots, x_m) = [\varphi](ax_1, \ldots, ax_m) \qquad and \qquad R_\varphi = Q_\varphi \cdot (1 + S_\varphi).$$

Let t be a family of values of type $s_1(\varphi)$, let F be a 3-path with 2-source $t \star_1 \varphi$, made of k computation 3-cells and l structure 3-cells. Then the following inequalities hold:

$$k \leq Q_\varphi(\|t^1\|, \ldots, \|t^m\|)) \qquad and \qquad l \leq Q_\varphi(\|t^1\|, \ldots, \|t^m\|)) \cdot S_\varphi(\|t^1\|, \ldots, \|t^m\|)).$$

As a consequence, $\||F\|| \leq R_\varphi(\|t^1\|, \ldots, \|t^m\|)$ holds.

Proof. We decompose F into a $\star_2$-composite of elementary computation 3-paths followed by structure 3-paths. Using the fact that the heat map we consider is strictly decreasing on computation 3-cells and weakly decreasing on structure 3-cells, we deduce that $[t \star_1 \varphi]$ is minored by k. We use the properties of $[\cdot]$ and lemma 4.2 to get the bound we seek on k. Then, we apply proposition 4.7 to each of the structure 3-paths we have isolated. We sum up the resulting inequalities and use the facts that $[t \star_1 \varphi]_S = 0$ and $[t_2(F)]_S \geq 0$ to get $k \cdot S_\varphi(\|t^1\|, \ldots, \|t^m\|) \geq l$. We deduce the inequality on l from this one and the one on k. We conclude by using the equality $\||F\|| = k + l$. $\qquad \square$

Example 4.8. For the functions of example 2.6, we have $Q_\bullet(x) = 2x^2$, $Q_\blacktriangle(x) = x$ and $Q_\blacktriangledown(x, y) = x + y$. For example, let us fix a list t. The polynomial $Q_\bullet$ tells us that, during the computation of the sorted list $\bullet(t)$, there will be at most $\|t\|$ applications of a computation 3-cell. The polynomial $R_\bullet$ guarantees that there is no more than $\|t\|^2(1 + \|t\|^2)$ applications of rules. On the examples we have considered, the polynomial Q_φ gives a bound that is close to known ones but the polynomial R_φ gives a very overestimated bound. To get a better estimation, we will have to work on the structure heat increase bound S_φ.

Theorem 4.9. *Functions computed by simple confluent polygraphic programs are exactly* PTIME *functions.*

Proof. We start by proving that functions computed by simple polygraphic programs are in PTIME. Proposition 4.7 tells us that the length of any computation in such a polygraph are polynomially bounded by the size of the arguments. Furthermore, each step of computation can be done in polynomial time

with respect to the size of the current 2-path: we find a redex in a directed acyclic graph with decorations then replace it by the corresponding reduce and both operations can be done in polynomial time.

Now let us prove that any PTIME function can be computed by a simple polygraphic program. The first step is to translate a Turing machine equipped with a clock into a polygraphic program. We fix a function f in PTIME, a Turing machine $\mathcal{M}$ that computes f and a polynomial P that bounds the length of the computation. We consider a copy of the polygraphic program of example 2.4 which computes addition and multiplication of natural numbers, with its 1-source denoted by nat. Let us note that this polygraphic program computes any polynomials, including P. Then we extend it with a variant of the polygraphic Turing machine of section 2: it is made of a 1-cell mon; its constructors are the empty word $\varphi : \text{mon} \Rightarrow \text{mon}$, plus one cell $\phi : \text{mon} \Rightarrow \text{mon}$ for each letter of the alphabet of $\mathcal{M}$; its functions are the main $\bullet : \text{mon} \Rightarrow \text{mon}$ for f, plus a size function $\blacksquare : \text{mon} \Rightarrow \text{nat}$, plus a modified cell $\boxed{q\ a} : \text{nat} \star_0 \text{mon} \star_0 \text{mon} \Rightarrow \text{mon}$ for each state q of $\mathcal{M}$ and each letter a in the alphabet of $\mathcal{M}$, including the blank symbol $\sharp$; its computation 3-cells are:

 both when $\delta(q, a) = (q', c, L)$

 both when $\delta(q, a) = (q', c, R)$

Then, one checks that this polygraphic program mimics the transition of the original Turing machine $\mathcal{M}$ and, thus, computes f. We conclude by checking that the following polygraphic interpretation, extending the one already built on natural numbers, is simple and compatible with each computation 3-cell:

- $\varphi_* = 1$, $\phi_*(i) = i + 1$, $\blacksquare_*(i) = i$, $\boxed{q\ a}_*(i, j, k) = i + j + k$, $\bullet_*(i) = P_*(i) + i + 1$.

- $[\blacksquare](i) = i$, $\left[\boxed{q\ a}\right](i, j, k) = i$, $[\bullet](i) = [P](i) + P_*(i) + i + 1$.

$\square$

Actually, as the simulation is done step by step, another theorem follows, still taking the definition of NPTIME functions to be the one of Grädel and Gurevich [5]:

Theorem 4.10. *Functions computed by simple non-confluent polygraphic programs are exactly* NPTIME *functions.*

References

[1] Guillaume Bonfante, Adam Cichon, Jean-Yves Marion, and Hélène Touzet, *Algorithms with polynomial interpretation termination proofs*, Journal of Functional Programming **11** (2001), no. 1, 33–53.

[2] Guillaume Bonfante and Yves Guiraud, *Programs as polygraphs: computability and complexity*, Logical Methods in Computer Science (in press), 2006.

REFERENCES

[3] Albert Burroni, *Higher-dimensional word problems with applications to equational logic*, Theoretical Computer Science **115** (1993), no. 1, 43–62.

[4] Adam Cichon and Pierre Lescanne, *Polynomial interpretations and the complexity of algorithms*, Lecture Notes in Artificial Intelligence **607** (1992), 139–147.

[5] Erich Grädel and yuri Gurevich, *Tailoring recursion for complexity*, Journal of Symbolic Logic **60** (1995), no. 3, 952–969.

[6] Yves Guiraud, *Termination orders for 3-dimensional rewriting*, Journal of Pure and Applied Algebra **207** (2006), no. 2, 341–371.

[7] _____ , *The three dimensions of proofs*, Annals of Pure and Applied Logic **141** (2006), no. 1-2, 266–295.

[8] _____ , *Polygraphs for termination of left-linear term rewriting systems*, Preprint, 2007.

[9] Yves Lafont, *Interaction nets*, Principles of Programming Languages, ACM Press, 1990, pp. 95–108.

[10] _____ , *Towards an algebraic theory of boolean circuits*, Journal of Pure and Applied Algebra **184** (2003), no. 2-3, 257–310.

[11] _____ , *Algebra and geometry of rewriting*, To appear in Applied Categorical Structures, 2006.

[12] Yves Lafont and François Métayer, *Polygraphic resolutions and homology of monoids*, Preprint IML, 2006.

[13] Dallas Lankford, *On proving term rewriting systems are noetherian*, Tech. report, Louisiana Tech University, 1979.

[14] François Métayer, *Resolutions by polygraphs*, Theory and Applications of Categories **11** (2003), 148–184.

[15] Detlef Plump, *Term graph rewriting*, Handbook of Graph Grammars and Computing by Graph Transformation **2** (1999), 3–61.

[16] John von Neumann, *Theory of self-reproducing automata*, University of Illinois Press, 1966.